

本周周报(7.23-7.29):

解聪

本周工作:

1. 交易地图

本周的交易地图工作包括两个部分: 利用 OpenCL 并行实现 KDE 算法部分, 以及动态累计生成交易背景密度图。

1) .使用 GPU 加速 KDE 的计算。

环境配置

先前的计划是使用 CUDA 做并行计算。由于公司的计算机使用 ATI 显卡, 因此更改方案, 使用 OpenCL 代替 CUDA。

在配置 OpenCL 环境的时候遇到了不少问题。ATI 显卡的计算机要使用 OpenCL, 需要首先安装 ATI Stream SDK。在安装后发现程序始终有错误提示。因为公司显卡版本太旧, 而且公司使用的 XP 操作系统是新版 SDK 不兼容的, 所以有更换了较早的版本的 ATI Stream SDK。

因为程序采用 Java 和 Processing 编写。Java 平台下不能直接使用 OpenCL, 因此使用了 OpenCL 的 java 绑定的库-JOCL。

算法实现

KDE 的并行计算参考了论文“Interactive Visualization of Streaming Data with Kernel Density Estimation”。

文中总结估计 2D KDE 的方法可以分为如下两类:

对于密度场中某一个像素单元 c, v 和 k 是其密度值与位置。 K 是 2D 平面上的散点样本。

- a) for c in cells:
 - for k in kernels:
 - $c.v += k.eval(c.p)$
- b) for k in kernels:
 - for c in cells:
 - $c.v += k.eval(c.p)$

a) 首先针对每一个像素进行循环, 计算出不同的核在该像素点上的累积值;
b) 首先针对密度核进行循环, 计算该样本点对每个像素点的和密度的累积。
这里采用 a) 方法。因为 a) 方法利于并行, 即利用每个运算单元分别计算 2D 密度场中的每一点像素值。

与文章中方法不同的是, 由于本程序在 Java 平台上运行, 所以密度场直接使用了二维数组保存。而参考论文中将密度场做成纹理。并使用了 2D 的 FBO。至于具体实现的细节, 此处就不细说了, 因为 OpenCL 是初次使用, 所以在实现过程中同样遇到了不小问题, 花费了不少的时间。

效率分析与问题

使用了 OpenCL 进行加速并测量了两种方法的计算时间差异。对于程序中同

样的一组输入数据，其分辨率为 800*900，使用两组核函数进行计算并将其聚合。直接使用 CPU 计算则需要 43 秒，而使用了 OpenCL 并行计算的程序总的运行时间大约为 1 秒左右。因此从计算效率的角度来讲，性能的提升是十分明显的。

在使用并行计算尝试动态更新背景密度图时，发现在更新背景密度图的时候还是比较卡。在经过一系列的测试以后，发现程序效率的瓶颈出现在 IO 方面。主要问题在于，OpenCL 在分配以及读/写共享内存时使用了指针，在 C 的环境下就是传共享内存的地址，其效率比较高。而 Java 中并不能直接使用指针对存储区域进行操作，JOCL 采用了 Pointer 这样的伪指针来替代。而每次更新密度图，计算 KDE 时都需要进行共享存储的分配，读写与回收。采用 Java 的方式导致读写效率降低了许多。虽然程序的计算性能的确是得到了提升，但是程序的读写速度成为了运行效率的瓶颈。

2) .动态更新背景的密度图

先前实现中，密度图的效果是预先生成的静态背景，在程序运行过程中是不会改变的。本周的工作对此进行改进，加进了随着时间动态改变密度图的功能。动态背景的生成同样参考了论文“Interactive Visualization of Streaming Data with Kernel Density Estimation”。

论文对于动态数据的处理与我们先前的想法类似，同样是设置了三个变量：当前时刻，时间窗口与时间步长。在实现过程中，时间窗口取了一个小时（即统计前一个小时的交易信息），时间步长取了 2 秒钟（即每两秒更新一次）。

动态更新实现的主要步骤如下：

1. 由当前时刻计算出时间窗口的位置。统计出时间窗口内的交易信息。（由于原始数据本来就是按照一个小时统计好的，所以这里只能用随机生成的伪造数据）特别地，若当前时刻为 0，则交易量从 0 开始累积。
2. 由统计得到的时间窗口内的交易信息，使用不同的核函数计算当前的密度图，并对两种核估计出来的密度图予以聚合。
3. 使用密度函数作为高度场，用 Phong 模型加上光照。先将计算得到的每个像素点的 RGB 值绘制到图像缓存中。最后绘制在屏幕上。
4. 判断当前时刻距上次更新是否达到了时间步长，若达到，则更新计算密度图。重复步骤 1-3。

图 1 显示的是从 0 开始累积的交易密度图。

与先前的效果不同的是，本周修改了程序，在配色方案上选取了单色（图 1 采用颜色较偏蓝色）。因为考虑到颜色可能会用来编码其他信息，比如：类目信息；另外玄澄表示采用配色方案使得颜色很杂乱，推荐就选择单色。添加了城市名与时间的文本信息。以后还会进一步完善相关真实信息，比如加入地理信息。

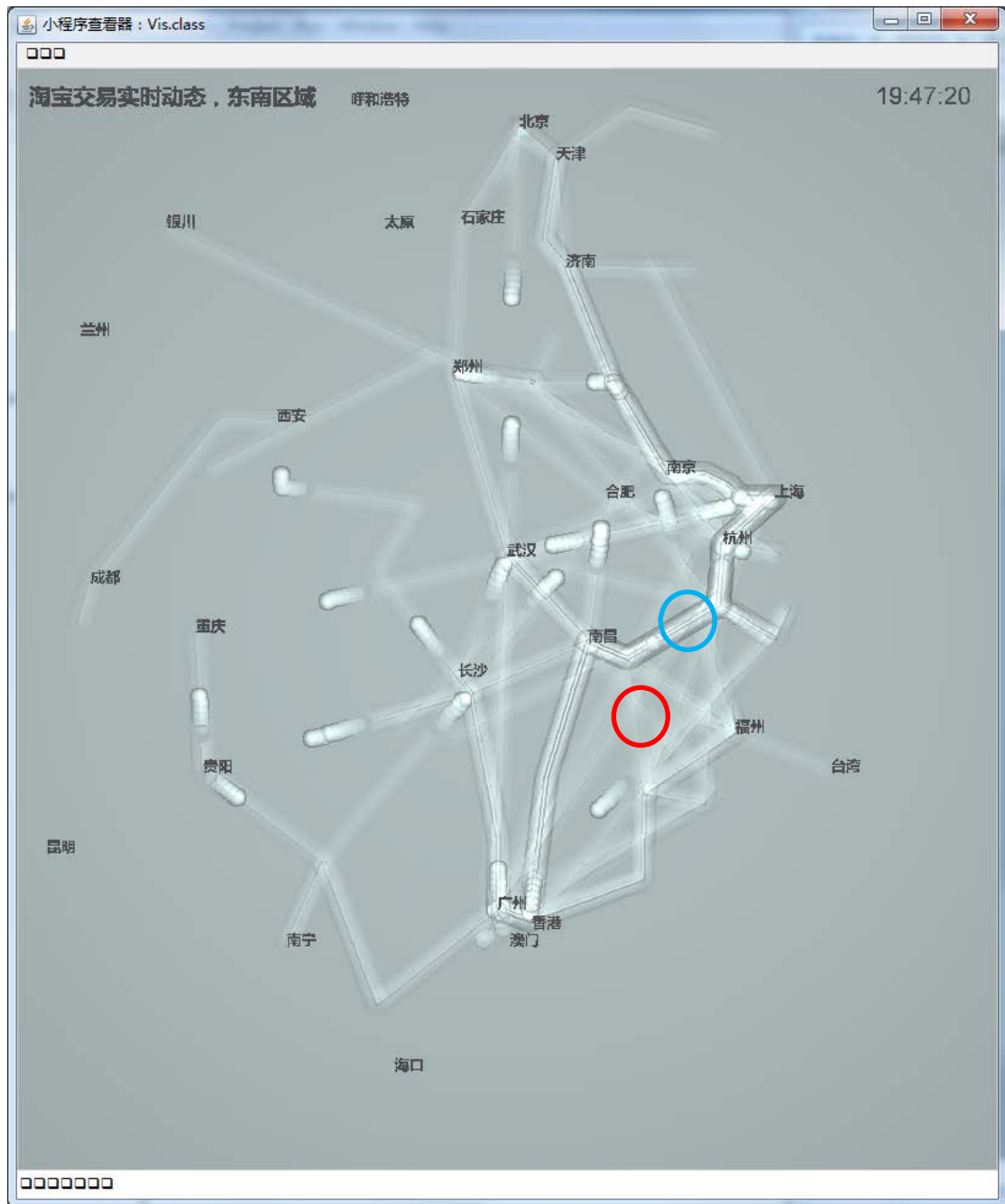


图 2 程序运行 20 秒背景密度图效果

图 2 显示在 20 秒过后, 交易的累积量渐渐变大。对于公式 1 中的 n 值增加, 更新后的部分像素点出密度值就逐渐减小。在图中表现出来就是部分交易渐渐变淡, 比如红色所示。同时, 部分交易权值依然较大, 在图中依然很明显, 比如蓝色部分所示。原因在于此处的交易一直保持密集, 更新后此处的交易值随着总交易量变大。

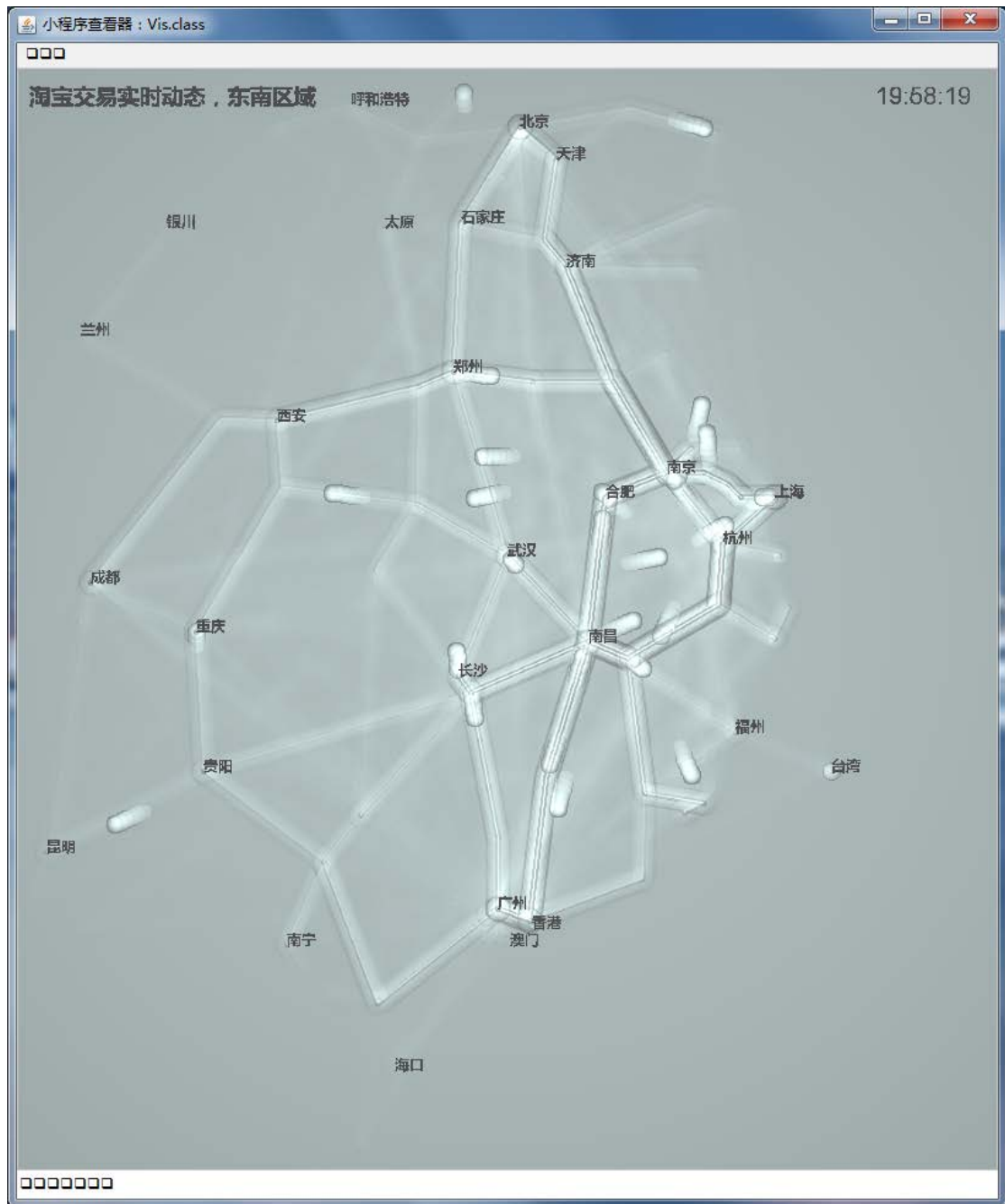


图 3 十分钟后的背景交易图。交易图渐渐趋于稳定。

图 3 显示 10 分钟过后总的样本个数比较大, 个别样本点的增加与减少对全局的密度估计结果的改变已经是很微小了。整个交易的密度图趋于稳定, 单笔交易的增加对整个密度图影像不大。密度图此刻仅仅存在局部的微小更新。可以看出几条主要干线, 比如北京到上海, 上海到广州的铁路线。为了加速运算, 在实现过程中, 对于全局的密度图变化不会太大的情况, 有时只存在局部的密度图的变化。因此可以减小计算量, 只局部的密度图采用 KDE 重新计算、更新光照即可, 无需再对全局的密度图重新计算。

录制交易地图背景动态更新的视频, 并上传到 FTP, 地址为:

ftp://10.76.0.157/Incoming/Users/xiecong/taobao_7_29.swf

2. 组件库工作

本周组件库主要集中在展示与交互的改进。包括力引导布局组件以及先前李逢博负责的弦图的组件改进。

力引导布局的改进主要包括鼠标单击事件的改进，添加了当鼠标单击在某点上，显示其关联节点的效果；添加了权值较重节点的文字跟随。

对布局参数做了修改，使布局分布较为松散，扩大了节点的半径。

弦图的修改主要在于显示效果方面。添加了当鼠标悬浮在某条弦上，其他无关弦隐掉的效果。外观上调整了边框等设计。

3. 教材编写

将第九章的五节内容整合，并且按照第一章调整了格式。包括引文格式，以及图片题注格式。

对每个小节内容调整了分类的方式并且对每一小节做了概述与总结。

下周工作：

1. 淘宝交易地图

下一步的工作包括如下几个部分：同一条路径上不同方向的可视化，同城信息的可视化，密度图与真实地理信息的融合，二维平面到球面的映射。

对于第一个问题，之前有过几次简单的尝试，比如颜色编码不同方向，或者密度不同编码异向信息。现在还没有十分合适的方法。

对于第二个问题目前是比较明确的。同城的问题的解决需要在每个一级城市位置放置可视化工具。而且同城可视化既可以反映及时交易，也可以反映本城市的累积交易信息。

第三个与第四问题是在完善好目前方案的基础上做的，对程序是一个提升，也可能会涉及到设计方面的问题。

2. DataV 组件库

继续按照设计人员讨论后的方案修改组件库中力引导布局与弦图的交互与外观。准备发布组件前的一些工作。

3. 教材整理。